★ ACTIONQA HORROR COMICS ★ NO. 1 ★ 2026 ★

FREE BUILD SPEC!

Your fiend-ly QA host dares you to read…

BUILDING AN AGENTIC OS

A TALE OF MACHINE MEMORY & THE THINGS THAT REMEMBER

The companion build-spec to the videos — research-backed, tool-agnostic, and horribly easy to follow.

MEMORY · KNOWLEDGE · ENFORCEMENT

ACTIONQA · v2 · 2026-06 · research synthesized 2026-05-19 + 2026-06-02

HOW TO READ THIS TALE

Most of what follows is **portable** — the same files and ideas work on any operating system and with any coding agent. Where something is *not* portable, a badge tells you so, and an alternative is always given.

Just as important: this document keeps an honest line between **what the research says** and **what I decided myself**. Every meaningful claim carries a marker. If it's a marker with an outline, it comes from a source you can check. If it's the filled blood-red ✦ MY CHOICE stamp, it's a design decision or opinion of mine — useful, but not peer-reviewed truth.

PORTABILITY BADGES

OS-SPECIFIC — Depends on your operating system or filesystem (e.g. Linux + btrfs). A cross-platform alternative is always provided.

AGENT-SPECIFIC — Depends on your coding agent (paths, how skills load). Shown here for **Claude Code**; adapt the path for Cursor, Codex, OpenCode, etc.

EVIDENCE MARKERS — WHERE A CLAIM COMES FROM

● PEER-REVIEWED — Published at a peer-reviewed academic venue (COLM, UIST, TMLR, ICLR).

◖ PREPRINT — Rigorous research on arXiv, *not yet* peer-reviewed — but cited by peer-reviewed work and backed by shipping systems.

◆ INDUSTRY / STANDARD — Frontier-lab engineering docs or an adopted open standard. Authoritative, not academic.

○ VISION — An influential talk or essay — a way of thinking, not a measured result.

✦ MY CHOICE — **Mine.** A design decision, simplification, or opinion I'm making — clearly separated from the sources above.

Every source sits in Appendix C with its venue and a link, numbered [1]–[18].

◆ ▲ ◆ ▲ ◆

WHAT YOU'LL BUILD

A persistent **memory and knowledge layer** that lives *outside* any single tool, so your coding agent stops waking up amnesiac. You tell it something once; it remembers across sessions — and even across different agents. The whole thing is plain Markdown files plus one small skill.

```
# The finished layout — plain files, no database
~/.aios/
├── AGENTS.md              # the harness: who you are + how memory works
├── HARNESS.md             # the long-form procedure (read on demand)
├── memory/                # EPISODIC — small, loaded every session
│   ├── user-*.md              #   facts about you
│   ├── feedback-*.md          #   how you want the agent to behave
│   └── session-*.md           #   one record per session (a recall index)
├── knowledge-base/        # SEMANTIC — grows large, loaded on demand
│   ├── README.md              #   conventions (the only KB file always read)
│   └── *.md                   #   cross-linked notes with [[wikilinks]]
├── skills/write-note/     # the ONE moving part: routes + writes notes
└── scripts/               # the safety net (snapshots)
```

Two stores, one skill, a harness file, and a backup script. That's the entire system.

Time: ~30–45 minutes. **You need:** a coding agent that loads a project/instructions file and supports "skills" or reusable prompts (Claude Code, Cursor, Codex, OpenCode…), plus a terminal. **You don't need:** Obsidian, a vector database, or a 15-question "identity interview."

✦ MY CHOICE

CONTENTS

SECTIONS

APPENDICES

SECTION A · THE IDEA

Every time you open a new chat, your AI coding assistant starts from zero. You re-explain who you are, what you're working on, what you already decided — every single time. The fix is to give it a memory that lives *outside* the tool: a small, well-shaped layer on disk that any agent can read at the start of every session. That layer is what people mean by an "agentic OS."

⚠ "BUT MY AGENT HAS MEMORY NOW" True — Claude Code shipped **Auto Memory**, and it helps. But it's *single-vendor* (locked to one tool) and it **doesn't survive compaction**: when the context fills and the chat is summarized away, what it "knew" goes with it. A layer that lives *outside* the tool fixes both — and the enforcement hook later makes it survive the purge. ◆ INDUSTRY

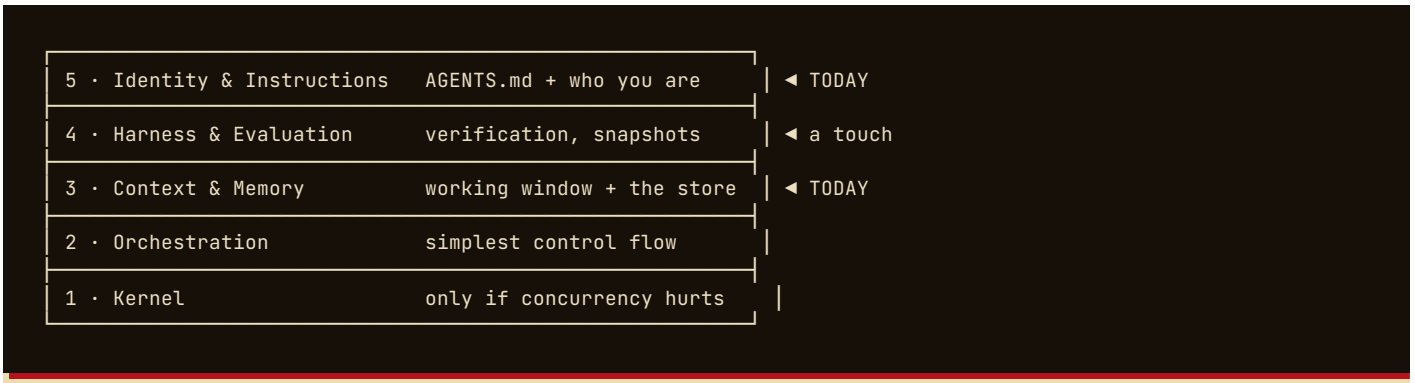
TWO MEANINGS OF "AGENTIC OS" — YOU ONLY NEED ONE

The phrase fuses two different things, and conflating them is where most projects over-build:

THE METAPHOR	THE LITERAL KERNEL
Karpathy's "LLM OS." Treat the model as a *kernel* that orchestrates memory, tools, storage, and I/O — the context window is RAM, an external store is disk. A *way of thinking* about a single agent. ○ VISION [1]	**The AIOS paper.** A real scheduler that isolates and time-slices resources for *many concurrent agents* on shared hardware. Actual infrastructure. ● PEER-REVIEWED [2]
For a personal build on your own machine, you only need the metaphor. If your model is a cloud API there's no scarce local resource to schedule; if it's a local GPU, a plain one-at-a-time queue solves it. The literal kernel is for multi-user platforms — explicitly out of scope here. ● [2] ✦ MY SCOPE	

THE FIVE-LAYER MAP — AND WHAT WE BUILD TODAY

The full reference architecture has five layers. This guide builds the parts that give you the biggest payoff first: **Layer 3 (memory)** and **Layer 5 (instructions/identity)**, with a touch of Layer 4 (the safety harness). The rest are noted so you know where today's work fits.



The guiding rule at every layer: start with the simplest thing; add complexity only when a concrete problem forces it. INDUSTRY [3]

THREE DESIGN CHOICES BEHIND THIS BUILD

- **Two memories, split by function.** Cognitive-science research splits long-term memory into *episodic* (what happened) and *semantic* (distilled facts) — different contents, different lifecycles, so don't pool them. PEER-REVIEWED [8][7] Turning that into *two folders* (`memory/` vs `knowledge-base/`) is my implementation of it. MY CHOICE
- **Plain files, not a database.** The shipping pattern from Anthropic's memory tool is a client-side file directory the agent reads first, assuming its context may have been reset — no vector DB required to begin. INDUSTRY [10]
- **Minimal-first, architecture-first.** Get a working skeleton, then personalize. No Obsidian, no identity interview, *one* skill instead of five. This is a deliberate opinion — I'm a test engineer, and even I say: prototype to learn the shape first, then bake in quality. MY CHOICE (it echoes Anthropic's "start simplest" rule [3])

SECTION B · THE BUILD

Now the hands-on part. Each step is copy-paste, with a one-line check so you know it worked. Commands assume a Unix-like shell (Linux/macOS/WSL). Where a step depends on your OS or your agent, you'll see a badge and an alternative.

⚠ BEFORE YOU START Don't put `~/.aios` inside a cloud-sync folder (Dropbox, iCloud, OneDrive). Sync conflicts will corrupt notes mid-write, and synced history is a poor place for anything sensitive. Keep it in your home directory.

0 LAYOUT & PREREQUISITES

Create the root and the two stores. A plain directory works everywhere; the btrfs line is an optional optimization that makes Step 6's snapshots instant.

```
# 1) Create the root.
mkdir -p ~/.aios

# OS-SPECIFIC alternative (btrfs only): a subvolume enables native snapshots in Step 6.
#   btrfs subvolume create ~/.aios

# 2) Create the stores + scratch + snapshot target.
mkdir -p ~/.aios/{memory,knowledge-base,skills/write-note,scripts}
mkdir -p ~/.aios-snapshots ~/.cache/aios
```

OS-SPECIFIC The `btrfs subvolume` line only applies to the Linux btrfs filesystem. On anything else, the plain `mkdir` is all you need — Step 6 gives you a universal snapshot option that works everywhere.

Check: `find ~/.aios -maxdepth 2` lists the four sub-folders.

1 THE TWO STORES

The split is the heart of the design. They differ by *function*, not format: ● PEER-REVIEWED [8]

STORE	HOLDS	LOADED	SIZE
`memory/` — **episodic**	Profile facts + one record per past session	Every session start	Must stay small
`knowledge-base/` — **semantic**	Distilled, reusable knowledge that compounds	On demand	Can grow large

Each note is a Markdown file with a little YAML frontmatter. A `memory/` note:

```
---
name: prefers-concise-answers
description: User prefers concise, answer-first responses with no filler
type: feedback
captured: 2026-05-23
---

Lead with the answer, not the reasoning. Skip filler and trailing summaries.

**Why:** the user reads fast and finds preamble noise.
**How to apply:** open with the result; add reasoning only if it changes what to do next.
```

The five note `type`s — one judgement about what a note *is*:

TYPE	WHAT IT IS	READ AT SESSION START?
`user`	Facts about you (work, context, tastes)	Yes — full body
`feedback`	How you want the agent to behave (rule + why)	Yes — full body
`project`	Project state & decisions — tag `status: active|archived`	`active` → full body; `archived` → one-line index only
`reference`	Pointer to one of your external systems	Yes
`session`	One compact record per session	No — only its `description:` is a recall index

The `name/description/type/captured` shape is mine, kept deliberately minimal so it stays vendor-neutral. ✦ MY CHOICE
The "small always-resident core, page the rest in on demand" principle behind it is from MemGPT. ◖ PREPRINT [9]

Why `status:`? Without it, *every* project note loads in full every session — including long-finished work. The first real run of the enforcement hook compiled a **35 KB** profile that way. `status: archived` is the deterministic stand-in for the prose's vague "active projects": only active ones load in full; the rest drop to a one-line index. ✦ MY CHOICE

2 THE ONE SKILL: `write-note`

The system has exactly one moving part: a skill that decides *where* a note belongs and writes it. (My first draft had five skills. One does the job — fewer moving parts, less to break.) ✦ MY CHOICE

Create `~/.aios/skills/write-note/SKILL.md`. Its own frontmatter stays portable — just a name and a trigger-rich description so the agent knows when to fire it:

```
---
name: write-note
description: Use when a durable fact should outlive the session — a fact about
  the user, a binding preference, active project state, or a synthesized learning.
  Persists it to the AIOS store.
---
```

The skill body tells the agent its procedure. The essentials, in order:

1. **Secrets check first.** Refuse to write if the text matches a credential pattern. This is the one rule that matters regardless of OS — a leaked secret is sticky in *any* history. ✦ MY CHOICE
2. **Pick destination + type** — one judgement: is this an episodic fact/profile (→ `memory/`) or a distilled, reusable concept (→ `knowledge-base/`)?
3. **Grep before creating** — if a note on the concept exists, extend it in place; only create a new file when the concept is genuinely new. (Exception: `session` records are always new.)
4. **Stamp** `captured:` on create and substantive edits (not typo fixes).
5. **For KB notes, link** — add `tags:` and `[[wikilinks]]` to neighbours.

The secrets patterns (a backstop — you are the first line):

```
# refuse the write if either matches
sk-[A-Za-z0-9-]{20,}                          # API keys (sk-ant-, sk-proj-, …)
-----BEGIN (RSA |EC |OPENSSH |)?PRIVATE KEY-----  # private keys
```

Why no "save this?" prompt? The agent writes memory *without* asking each time — otherwise you get approval fatigue and it stops bothering to remember. That's only safe because Step 6 gives you a time machine to undo any bad write. The mechanism (memory edits as ordinary actions) is MemGPT ◖ PREPRINT [9]; the conservative, user-inspectable posture is Anthropic's ◆ INDUSTRY [10]; the "no prompt, rely on snapshots" trade is mine. ✦ MY CHOICE

A complete, paste-ready `SKILL.md` is in Appendix A.

3 KNOWLEDGE-BASE CONVENTIONS

The KB is the long-term half — think a personal wiki crossed with a Zettelkasten: one concept per file, every note cross-linked so a thread can be followed later. ✦ MY CHOICE Only one KB file is read every session — `README.md` — and it describes *conventions, not contents*. This is the "navigation index" pattern: a curated map the agent reads first to orient itself. ◆ STANDARD [14]

A KB note adds `tags:` and links:

```
---
name: episodic-vs-semantic-memory
description: Why AIOS keeps two stores — episodic memory vs semantic knowledge
type: reference
tags: [memory, architecture]
captured: 2026-05-23
---

AIOS splits `memory/` and `knowledge-base/` by *function*, not format — the
classic episodic-vs-semantic distinction. Both are written by [[write-note]].
```

Finding notes later is just shell — no index to maintain:

```
grep -l "tags:.*memory" ~/.aios/knowledge-base/*.md   # by tag
grep -l "\[\[write-note\]\]" ~/.aios/knowledge-base/*.md  # backlinks
```

4 WIRE THE BRAIN: `AGENTS.md`

Now the agent needs to know the store exists and how to use it. Put that in one canonical instructions file. Use the **AGENTS.md** standard — it's cross-vendor (60k+ projects), so the same file serves every tool. ◆ STANDARD [6] Keep it lean: bloated instruction files get ignored. ◆ INDUSTRY [6]

Write `~/.aios/AGENTS.md` with two parts — your preferences, then the memory procedure (the *harness loop*):

```
# Personal Preferences
- Be concise. Lead with the answer, not the reasoning.
- Don't refactor untouched code. Avoid over-engineering.

# AIOS memory
Store at ~/.aios/: memory/ (always-on profile + session records) and
knowledge-base/ (synthesized notes). All writes go through write-note —
no approval prompt; never persist secrets.

- Session start · read memory/ notes (feedback, user, active project);
  treat each session note's description as a recall index. Read knowledge-base/README.md.
- During · capture durable facts the moment they surface; recall a past
  session from the index when the user refers back to earlier work.
- Session end · write one compact session record + distil learnings to the KB.
- Boundaries · durable state under ~/.aios/, ephemera under ~/.cache/aios/.
```

The write / retrieve / **consolidate** lifecycle (synthesize higher-level notes from clusters of raw ones) is from Generative Agents ● PEER-REVIEWED [7]; reading the store first "assuming context may have been reset" is Anthropic's memory tool ◆ INDUSTRY [10]. The exact three-phase wording is mine. ✦ MY CHOICE

> ⚠ **PROSE IS A WISH, NOT A GUARANTEE** This session-start instruction *works*, but the agent can skip it — and **compaction will wipe** whatever it loaded. The enforcement hook replaces this prose load with a deterministic one; the reconciled wording ("injected by the hook — don't re-glob") is what ships in Appendix A2. Until you add the hook, this prose load is your working v1.

Now point your agent at it. **This part is agent-specific** — shown for Claude Code:

```
# GUARDRAIL 1 — copy your existing skills in, then VERIFY (only write-note should be extra)
cp -an ~/.claude/skills/. ~/.aios/skills/
diff <(cd ~/.claude/skills && ls -A) <(cd ~/.aios/skills && ls -A)

# GUARDRAIL 2 — back up the originals (rename, don't delete)
mv ~/.claude/CLAUDE.md  ~/.claude/CLAUDE.md.pre-aios.bak
mv ~/.claude/skills     ~/.claude/skills.pre-aios.bak

# FLIP — symlink the agent's config to the harness
ln -s ~/.aios/AGENTS.md ~/.claude/CLAUDE.md
ln -s ~/.aios/skills    ~/.claude/skills
```

AGENT-SPECIFIC The paths (`~/.claude/CLAUDE.md`, `~/.claude/skills`) are Claude Code's. For another agent, symlink *its* instructions file to `~/.aios/AGENTS.md` and point it at the skills folder however it loads reusable prompts. The `~/.aios/` content never changes — only the wiring does. Because `AGENTS.md` is a shared standard, one file can serve several agents at once.

Rollback (keep this handy until you trust it):

```
rm ~/.claude/CLAUDE.md ~/.claude/skills \
  && mv ~/.claude/CLAUDE.md.pre-aios.bak ~/.claude/CLAUDE.md \
  && mv ~/.claude/skills.pre-aios.bak ~/.claude/skills
```

5 WIRE THE SKILLS

Step 4 already symlinked the skills directory. Restart your agent and confirm it loaded the harness: it should pick up your preferences and list `write-note` among its skills. AGENT-SPECIFIC — how skills surface (slash-commands, a skills menu, etc.) varies by tool. The portable truth: the agent reads reusable prompts from a folder you control, and that folder is now `~/.aios/skills`.

Check: `ls -l ~/.claude/skills` resolves to `~/.aios/skills`, and a fresh session can run `write-note`.

6 THE SAFETY NET: SNAPSHOTS

Auto-write is only safe because you can travel back in time. This is the *one* layer that is genuinely OS-specific — so pick your tool and move on. Its only job: version `~/.aios/` on a timer and restore cheaply.

OPTION A · LINUX + BTRFS (WHAT THE VIDEO USED) OS-SPECIFIC

```
# ~/.aios/scripts/snapshot.sh
#!/usr/bin/env bash
set -euo pipefail
ts=$(date -Iminutes)
btrfs subvolume snapshot -r "$HOME/.aios" "$HOME/.aios-snapshots/$ts"
```

Run it every 15 minutes with a **systemd user timer** (no root needed):

```
cat > ~/.config/systemd/user/aios-snapshot.service <<'EOF'
[Unit]
Description=AIOS snapshot
[Service]
Type=oneshot
ExecStart=%h/.aios/scripts/snapshot.sh
EOF

cat > ~/.config/systemd/user/aios-snapshot.timer <<'EOF'
[Unit]
Description=AIOS snapshot every 15 min
[Timer]
OnBootSec=2min
OnUnitActiveSec=15min
Persistent=true
[Install]
WantedBy=timers.target
EOF

systemctl --user daemon-reload
systemctl --user enable --now aios-snapshot.timer
```

OPTION B · THE UNIVERSAL ONE — RESTIC (CROSS-PLATFORM) ✦ RECOMMENDED OFF-BTRFS

A real snapshot tool — deduplicated, incremental, and *encrypted at rest* — that behaves identically on Linux, macOS, and Windows. `restic init` sets a repo password; keep it somewhere safe (not inside `~/.aios`).

```
restic init -r ~/.aios-restic
# on a timer (cron · launchd · Task Scheduler):
restic -r ~/.aios-restic backup ~/.aios
# list snapshots:
restic -r ~/.aios-restic snapshots
# restore a snapshot, then copy the file back:
restic -r ~/.aios-restic restore <id> --target /tmp/restore
```

OPTION C · THE SIMPLEST ONE — GIT (ALREADY ON EVERY MACHINE)

```
git -C ~/.aios init -q
# on a timer (cron · launchd · Task Scheduler):
git -C ~/.aios add -A && git -C ~/.aios commit -qm "snapshot $(date)" || true
# restore one file:
git -C ~/.aios checkout <commit> -- memory/<slug>.md
```

YOUR SETUP	NATIVE OPTION	UNIVERSAL (RECOMMENDED)
Linux · btrfs	`btrfs subvolume snapshot`	restic / git
Linux · ext4/xfs	— none native —	restic / git
Linux · ZFS	`zfs snapshot`	restic / git
macOS · APFS	`tmutil localsnapshot`	restic / git
Windows · NTFS	File History / VSS	restic / git
WSL2 on Windows	— usually ext4 —	restic / git

⚠ SECRETS ARE STICKY A leaked secret lives forever in *any* history — btrfs snapshot, git log, or restic. That's exactly why the write-time secrets check in Step 2 matters no matter which backup tool you choose.

Check: trigger one snapshot, edit a note, then restore it from the snapshot. *Always test your backups.*

7 PROVE IT REMEMBERS

The payoff — does it actually remember you across a fresh context?

```
# 1) In a session running the wired harness, tell it something durable:
"Remember I build AI tooling, I'm on Linux, and I work solo."
#    → write-note fires → a new type:user file appears:
ls -lt ~/.aios/memory/

# 2) Open a BRAND-NEW session and ask:
"What do you know about me?"
#    → it answers from memory/ — across a fresh context.  ✦ the loop closes

# 3) Wrap up: "let's stop here." → end-of-session synthesis writes a
#    session record + a KB note.
```

Honest beat: end-of-session synthesis only fires if the agent *notices* the session ending. If it doesn't, just say "wrap up and save what we learned." v1 is a skeleton you grow — not a finished product. ✦ MY CHOICE

The real test (after the hook): once the enforcement hook is in, run `/compact` (or fill the context until it auto-compacts) and ask again. If the profile is still there, you've beaten the one failure prose loading can't. ✦ MY CHOICE

SECTION C · MAKING MEMORY STICK — THE HOOK

The build above works… until it doesn't — and when it fails, it fails *silently*. The whole layer rests on one prose instruction: "at session start, read memory/." Prose is a *request*, not a guarantee. The model can ignore it on any turn, and the moment your context fills and the platform **compacts** the conversation into a summary, everything it loaded is purged with the rest. No instruction re-fires. The profile vanishes and the agent works as if it never knew you. This chapter nails that load down — deterministically.

THE DETOUR: DON'T REACH FOR THE BIG TOOL YET

The obvious move is to grab an off-the-shelf memory tool — they promise hooks, vector recall, automatic consolidation, and decay for free. I tried one. It installed clean on one machine and **broke hard on another, mid-build** (an OS-specific install failure). So I stopped and did a research pass before rebuilding it myself — and the research said most of those features were premature at my scale: ✦ MY CHOICE

- **Vector / embedding recall was ~10× premature.** The whole corpus was ~36 notes — it fits in a single context window. `grep` is plenty; embeddings would drag in a fragile dependency for a payoff that rounds to zero at this scale. ◆ INDUSTRY
- **An LLM consolidation / decay engine** solves a scale problem 36 notes don't have — you can still eyeball them by hand.
- **No off-the-shelf tool fit cleanly** — the mature ones trade plain-text legibility for scale you don't have yet, and several wanted a background daemon or a store migration.
- **The platform now ships its own memory** — but it's single-vendor and doesn't survive compaction. The differentiator of a hand-rolled layer is being *cross-vendor*.

☠ THE LESSON, RE-LEARNED Don't over-engineer. Build the one thing the failure actually demonstrated; defer the rest behind a tripwire (below) that tells you when a deferral has expired — instead of guessing.

THE FIX: ONE CORE SCRIPT, THIN PER-VENDOR SHIMS

Move the load from "something the model is asked to do" to "something the harness does deterministically, every session and after every compaction." The architecture is **one vendor-neutral core + thin per-vendor adapters**:

- **The core** — a single compile script (e.g. `~/.aios/bin/aios-profile.sh`) prints the always-on slice to stdout: full bodies of feedback / user / *active* project notes, a one-line index of session notes & archived projects,

and a KB pointer. It parses frontmatter with a tiny inline `python3` block — **no new dependencies**.

✦ MY CHOICE

- **The adapters** — each agent platform just runs that script and injects its output at session start. Because all the logic lives in one script, every adapter is a three-line shim, not a rewrite. (Inlining the logic into one vendor's hook would lock you to that vendor.)

AGENT-SPECIFIC · CLAUDE CODE One `SessionStart` hook runs the script. The key fact: with **no matcher** it fires on `startup | resume | clear | compact` — so the *same hook re-injects the profile after compaction.* That is the exact failure this chapter exists to fix. Other agents: find the session-start hook and **verify it re-fires after compaction**; if not, inject once on the first message, guarded by a per-session flag file. A build-it-with-your-agent prompt: Appendix A5.

Then reconcile the prose. Once the hook injects deterministically, change the `AGENTS.md` session-start line so the model doesn't *also* glob memory/ — otherwise it loads twice. New wording: "the profile is injected automatically by the hook — do not re-glob; on-demand recall of a past session's body still applies." (Reflected in Appendix A2.)

THE TRIPWIRE: KNOW WHEN YOU'VE OUTGROWN SIMPLE

Deferring features is only responsible if something tells you when a deferral has expired. Since the compile script already reads every note, it cheaply measures a few escalation signals and prints a single ⚠ line **only when a threshold trips** (silent otherwise). You stop guessing "is it time for real search yet?" — the system tells you.

✦ MY CHOICE

DEFERRED FEATURE	TRIPS WHEN…
Ranked recall (SQLite FTS5, stdlib)	note count ≥ ~150
Embedding dedup (offline, in the synthesis pass)	note count ≥ ~300
Vector / hybrid recall	note count ≥ ~500 *and* the miss-log is non-empty
LLM consolidation / decay engine	compiled profile ≥ ~40 KB
Auto-capture hooks	a logged pattern of missed facts

All numbers are starting points, tunable in the script. A non-empty **recall-miss log overrides the counts** — it's evidence of a real miss, not a proxy.

Tune on real data — but never to hide a real signal. When the first compile hit 35 KB it was *genuine bloat* (finished projects loading in full); the fix was structural — the `status: archived` marker — not raising the limit. Later, legitimate preference notes pushed it to ~26 KB with nothing left to trim; *that* warranted raising the limit (to ~40 KB) with a written rationale. If the signal points at structural waste, fix the structure; if at a guessed-wrong constant, retune it and write down why.

✦ MY CHOICE

THE RECALL-MISS LOG

When recall fails — you *know* a note exists but search can't surface it — append one line:

```
echo "$(date +%F)  searched '<query>'  expected '<note/fact>'" \
  >> ~/.cache/aios/recall-misses.log
```

The tripwire counts non-empty lines. A filling log is the **strongest** "upgrade search now" signal — stronger than any corpus-size guess, because it's an actual miss, not a proxy.

SECTION D · MAKE IT YOURS

WHAT'S PORTABLE VS. WHAT TO CHANGE

COMPONENT	PORTABLE?	IF NOT, CHANGE...
The `memory/` & `knowledge-base/` files	**Fully portable**	—
The `write-note` skill body	**Fully portable**	—
`AGENTS.md` content	**Fully portable**	—
Wiring (symlink paths, skill loading)	AGENT	Point your agent's config/instructions path at `~/.aios/`
Snapshots	OS	Use restic/git instead of btrfs+systemd

COMMON MISTAKES (ALL SOURCE-BACKED)

- **Building the kernel first.** Layer 1 is the last thing you need, not the first. [2]
- **Dumping everything into context.** Bigger context ≠ better; curate it. [4]
- **Over-stuffing the instructions file** until the agent ignores it. [6]
- **Treating long-term memory as one bucket.** Episodic, semantic, and procedural have different lifecycles. [7][8]
- **No verification loop** — being the only thing that catches the agent's mistakes. [5]
- **Treating security as a later phase.** Prompt injection and data poisoning are design constraints from day one. [1] [2]

WHERE THIS GOES NEXT (FUTURE PARTS)

The skeleton plus the enforcement hook is v2. The rest stays deferred — but each now has an **automatic trigger** (the tripwire), so you're told when to build it instead of guessing:

- The **OpenCode adapter** (compaction re-inject + capture) — the next concrete cross-vendor step, a thin shim off the same compile script.
- Ranked → embedding-dedup → vector/hybrid recall, then an LLM consolidation / decay engine — each gated on a tripwire threshold, not a hunch. [7]
- `install.sh` / `uninstall.sh` so setup isn't manual.
- Evaluation: judge the agent at output / trajectory / regression scopes, and measure reliability (`pass^k`), not just one lucky run. [15] [16]

This guide covers the part that’s wired into v1. Future parts get built as I go — so grab updated specs when they ship.

APPENDIX A · COPY-PASTE TEMPLATES

A1 · write-note/SKILL.md

```
---
name: write-note
description: Use when a durable fact should outlive the session — a fact about
  the user, a binding preference, active project state, a pointer to an external
  system, or a synthesized learning. Persists it to the AIOS store.
---

# write-note

Persist a durable fact to ~/.aios. Procedure:

1. SECRETS CHECK FIRST. Refuse if the text matches:
   - sk-[A-Za-z0-9-]{20,}                          (API keys)
   - -----BEGIN (RSA |EC |OPENSSH |)?PRIVATE KEY-----  (private keys)
   Record that a secret *exists* if useful; never the value.

2. PICK DESTINATION + TYPE.
   memory/  → user | feedback | project | reference | session  (episodic + profile)
   knowledge-base/ → reference  (distilled, reusable concept)

3. GREP BEFORE CREATING. If a note on this concept exists, extend it in place.
   Create a new file only if the concept is genuinely new.
   (Exception: session records are always new: session-YYYY-MM-DD.)

4. FRONTMATTER: name, description, type, captured (stamp on create + real edits).
   KB notes also: tags: [..]  and [[wikilinks]] to neighbours.

5. WRITE with the editor's native file tools. No approval prompt — snapshots are
   the safety net. Boundaries: durable → ~/.aios/, ephemera → ~/.cache/aios/.
```

A2 · AGENTS.md SKELETON

```
# Personal Preferences
- Be concise. Lead with the answer, not the reasoning.
- No filler, no trailing summaries.
- Don't refactor untouched code. Avoid over-engineering.

# AIOS memory
Persistent store at ~/.aios/: memory/ (always-on profile + episodic session
records) and knowledge-base/ (synthesized, cross-linked notes). All writes go
through the write-note skill — no approval prompt; never persist secrets.

- Session start: the profile is injected automatically by the SessionStart hook —
  do NOT re-glob memory/. On-demand recall of a past session's body from the index
  still applies. Read knowledge-base/README.md.
- During: capture durable facts the moment they surface via write-note. Recall a
  past session from the index when the user refers back to earlier work.
- Session end: write one compact session record + distil learnings into the KB.
- Boundaries: durable state under ~/.aios/, ephemera under ~/.cache/aios/.
```

A3 · knowledge-base/README.md SKELETON

```
# Knowledge base — semantic, cross-linked
Synthesized, durable knowledge that compounds across projects. Read on demand.
This README is the only KB file preloaded each session.

## Finding notes
- By tag:   grep -l "tags:.*python" *.md
- Backlinks: grep -l "\[\[slug\]\]" *.md
- By text:  plain grep across bodies.

## Writing (via write-note)
- Grep before creating. Extend an existing note in place; new file only if new.
- Always add [[wikilinks]] to neighbours — an orphan note can't be followed later.
```

A4 · BTRFS SNAPSHOT SCRIPT OS-SPECIFIC

```
#!/usr/bin/env bash
# ~/.aios/scripts/snapshot.sh
set -euo pipefail
ts=$(date -Iminutes)
btrfs subvolume snapshot -r "$HOME/.aios" "$HOME/.aios-snapshots/$ts"
```

A5 · BUILD THE ENFORCEMENT HOOK WITH YOUR AGENT AGENT-SPECIFIC

Rather than hand-write the compile script and wire the hook yourself, have your own agent build both to a contract — so the always-on profile loads every session *and* survives compaction. Paste this:

```
Build me a memory-profile compile script + a session-start hook so my always-on
memory loads every session AND survives compaction. Steps — don't skip verify:

1. DISCOVER my store: markdown notes, one fact per file, YAML frontmatter. Report
   the frontmatter keys, the set of `type:` values, and where my knowledge base lives.
2. WRITE <store>/bin/aios-profile.sh (chmod +x) that prints the always-on slice to
   stdout — no new deps (use python3/awk):
     - FULL body of feedback / user / ACTIVE project notes (a project is active
       unless its frontmatter says status: archived);
     - ONE index line (captured · name: description) for session notes + archived
       projects; plus a KB pointer. Keep it compact.
     - TRIPWIRE: also measure note count, profile size in bytes, and a
       recall-miss-log line count, and print ONE ⚠ line only when a threshold
       trips (silent otherwise). A non-empty miss log overrides the counts. Put
       thresholds in labeled constants. Start: count ≥150 (ranked search),
       profile ≥~40 KB (consolidation), miss-log non-empty (better recall).
3. WIRE a session-start hook for THIS platform that injects the script's stdout on
   start AND after compaction (re-injection after compaction is the whole point).
   Claude Code: a no-matcher SessionStart hook piping through
   jq -Rs '{hookSpecificOutput:{hookEventName:"SessionStart",additionalContext:.}}'.
   Any other: find the session-start hook; VERIFY it re-fires after compaction; if
   not, inject once on the first message guarded by a per-session flag file.
4. RECONCILE my AGENTS.md: change "read memory/ at session start" to "injected by
   the hook — do not re-glob" (else it loads twice). Keep on-demand session recall.
5. VERIFY: run the script (bodies + index + KB pointer, compact); confirm the
   tripwire is silent now, then temporarily lower a threshold and confirm the ⚠
   appears. Then: a fresh session references a known fact without reading memory
   files, and the profile survives a /compact.
```

◆ ▲ ◆ ▲ ◆

APPENDIX B · CROSS-OS / CROSS-AGENT MAP

Everything in `~/.aios/` — the notes, the skill, `AGENTS.md` — is byte-identical on every OS and every agent. Only two things vary.

SNAPSHOTS, BY OS

OS / FS	NATIVE	UNIVERSAL	TIMER
Linux · btrfs	`btrfs subvolume snapshot`	restic / git	systemd / cron
Linux · ext4/xfs	—	restic / git	systemd / cron
macOS · APFS	`tmutil localsnapshot`	restic / git	launchd / cron
Windows · NTFS	File History (VSS)	restic / git	Task Scheduler

WIRING, BY AGENT

AGENT	POINT THIS AT `~/.aios/`
Claude Code	`~/.claude/CLAUDE.md` → `AGENTS.md`; `~/.claude/skills` → `skills/`
Cursor / Codex / others	Symlink the tool's instructions file to `AGENTS.md`; load reusable prompts from `skills/`
Multiple at once	Keep `AGENTS.md` canonical; symlink each tool's file to it

The cross-vendor instructions standard is AGENTS.md. ◆ STANDARD [6]

APPENDIX C · THE SOURCES

Everything in this guide traces to a primary source, vetted through a gaps → candidates → sources → synthesis pipeline. The promotion bar was strict: **primary, not a summary; a peer-reviewed paper or first-party frontier-lab engineering; architecture, not opinion.** Blog folklore and YouTube hot-takes were rejected.

✦ MY VETTING DISCIPLINE

EVIDENCE TIER	COUNT	WHAT IT MEANS
● PEER-REVIEWED	3 + 2	Published venues (3 core; SWE-bench & GAIA cited under [16])
◖ PREPRINT	3	arXiv, rigorous, cited by peer-reviewed work / shipping systems
◆ INDUSTRY / STANDARD	11	Frontier-lab engineering docs + adopted open standards
○ VISION	1	Karpathy's LLM-OS — influential framing

PEER-REVIEWED

[2] ● **AIOS: LLM Agent Operating System** — Mei et al., COLM 2025. arxiv.org/abs/2403.16971

[7] ● **Generative Agents** — Park et al., UIST '23 (memory stream: recency + importance + relevance, reflection). arxiv.org/abs/2304.03442

[8] ● **CoALA: Cognitive Architectures for Language Agents** — Sumers et al., TMLR 2024 (episodic/semantic/procedural taxonomy). arxiv.org/abs/2309.02427

[16+] ● **SWE-bench** (Jimenez et al., ICLR 2024, arXiv 2310.06770) & **GAIA** (Mialon et al., ICLR 2024, arXiv 2311.12983) — cited under τ-bench.

PREPRINT (ARXIV, NOT YET PEER-REVIEWED)

[9] ◖ **MemGPT** — Packer et al., 2023/24 (LLMs as OSes; self-editing memory; persona/human blocks). Successor: Letta. arxiv.org/abs/2310.08560

[11] ◖ **Constitutional AI** — Bai et al. (Anthropic), Dec 2022 (values as a designed artifact). arxiv.org/abs/2212.08073

[16] ◖ **τ-bench** — Yao et al., 2024 (trajectory grading; pass^k reliability). arxiv.org/abs/2406.12045

INDUSTRY / STANDARD

[3] ◆ **Building Effective Agents** — Anthropic. anthropic.com/engineering/building-effective-agents

[4] ◆ **Effective Context Engineering for AI Agents** — Anthropic. anthropic.com/engineering/effective-context-engineering-for-ai-agents

[5] ◆ **Claude Code Best Practices** — Anthropic. code.claude.com/docs/en/best-practices

[6] ◆ **AGENTS.md** — cross-harness instructions standard. agents.md

[10] ◆ **Memory Tool documentation** — Anthropic (files-first memory). platform.claude.com/docs/en/agents-and-tools/tool-use/memory-tool

[12] ◆ **Claude's Character / Constitution** — Anthropic. anthropic.com/research/claude-character

[13] ◆ **Introducing Contextual Retrieval** — Anthropic. anthropic.com/news/contextual-retrieval

[14] ◆ **llms.txt** — navigation-index standard (Howard, Answer.AI). llmstxt.org

[15] ◆ **Demystifying Evals for AI Agents** — Anthropic (output/trajectory/regression scopes). anthropic.com/engineering/demystifying-evals-for-ai-agents

[17] ◆ **OpenTelemetry GenAI semantic conventions** — observability spec. opentelemetry.io/docs/specs/semconv/gen-ai/

[18] ◆ **Citations API** — Anthropic (runtime grounding). platform.claude.com/docs/en/build-with-claude/citations

VISION

[1] ○ **"LLM OS" metaphor** — Andrej Karpathy, tweet (Sep 2023) + talk (Nov 2023). x.com/karpathy/status/1707437820045062561

APPENDIX D · QUICK-START CHECKLIST

The whole build on one page. Tick as you go.

- **0.** `mkdir -p ~/.aios/{memory,knowledge-base,skills/write-note,scripts}` & `~/.aios-snapshots`, `~/.cache/aios` — *not* in a cloud-sync folder.
- **1.** Understand the split: `memory/` = episodic (small, every session); `knowledge-base/` = semantic (grows, on demand).
- **2.** Write `skills/write-note/SKILL.md` — secrets check first, route by note type, grep before create. *(Appendix A1)*
- **3.** Write `knowledge-base/README.md` — conventions + grep idioms. *(Appendix A3)*
- **4.** Write `AGENTS.md` — preferences + the session-start/during/end loop. *(Appendix A2)*
- **4b.** Copy + verify skills, back up originals, symlink the agent's config → harness. Keep the rollback handy. AGENT
- **5.** Restart the agent; confirm preferences + `write-note` loaded.
- **6.** Set up snapshots on a 15-min timer (btrfs+systemd, or restic/git). **Test a restore.** OS
- **7.** Tell it a fact → open a fresh session → ask "what do you know about me?" → it remembers.

◆ ▲ ◆ ▲ ◆

HAPPY HAUNTING… ER, TESTING.

Built from the ActionQA "Agentic OS" research wiki · sources synthesized 2026-05-19 · This document: v2, 2026-06 (adds the enforcement hook). Future parts crawl out of the crypt as the build grows — grab updated specs when they land.